CSC108H Lecture 1

Dan Zingaro
OISE/UT

May 14, 2009

Welcome!

- Welcome to CSC108
- Comments or questions during class? Let me know!
- Previous experience: no programming experience!
- Topics: messing around with pictures, functions, conditionals, loops, debugging, testing, text file and webpage processing, searching, sorting, designing algorithms, and lots of other cool stuff!
- "Do I seriously have to buy the textbook?"
- Evaluation: four assignments, eleven labs, two tests, final exam, participation

Huh? I Can't Sleep in Class?

Well, you can, but:

- Participation is worth 7%
- But that's not the most important reason
- Learning is enhanced when we are actively engaged in and questioning the material
- You will have opportunities to discuss solutions to problems instead of just listening to me the whole time

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses
- 9. Disrespecting Other Students

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses
- 9. Disrespecting Other Students
- 8. Academic Offenses
- 7. Disrespecting Other Students
- 6. Academic Offenses
- 5. Disrespecting Other Students
- ...

How we will use Clickers

- An i¿clicker remote is required for this course (purchase through bookstore, register through `http://www.iclicker.com/registration`)
- I'll ask questions on the screen during lecture
- You can discuss with your neighbor and answer using your i¿clicker remote
- Class results are tallied, and I'll display a graph with the class results on the screen
- We'll discuss the questions and answers
- You get points for participating

How to Vote

- Turn on the clicker by pressing the bottom "On/Off" button.
- A blue "Power" light will appear at the top of the remote.
- When I ask a question in class (and start the timer), select A, B, C, D, or E as your vote.
- Green light: your vote was sent AND received; red light: vote again!
- You can always change your vote while the timer is going — your last vote is the one that counts

Clicker Tips

- If you bought a used clicker, replace all of the AAA batteries
- Do not use Duracell as they are TOO short for the casing.
- Do not use rechargeable batteries. They harm the clicker.
- Register your clicker before our next class
- Before using a new clicker for the first time, pull the plastic tab out of the battery compartment.
- Bring your clicker to class every day! Make sure your remote is on when voting!

Sample Clicker Question

How much programming experience do you have?

- A. Huh? What's programming?
- B. I programmed a little in high school
- C. I've programmed quite a bit before
- D. I'm very familiar with programming
- E. Out of the way Dan, I'm teaching this class!

What is Programming?

- A program is a set of instructions
- When you write down directions to your house for a friend, it is like a program that your friend "executes"
  - e.g. go forward three blocks, turn left at the light
- Computers have a different set of operations that they understand
  - Some are mathematical: add 10, take the square root
  - Others: read line from file, make pixel blue
- You can "teach" a computer new operations by defining them in terms of old ones
- Defining and combining new operations is the heart and soul of programming

Python History

- Late 1970s: programming language called ABC
  - High-level, intended for teaching
  - Only five data types
  - Programs are supposedly one-quarter the size of the equivalent BASIC or Pascal program
  - Not a successful project
  - More ABC information:
    `http://homepages.cwi.nl/~steven/abc/`

Python History...

- 1983: Guido van Rossum joined the ABC team
- Late 1980s: Guido needed a language for a different project; based it on ABC, removed warts (e.g. ABC wasn't extensible)
- Python, after Monty Python
- Guido: Benevolent Dictator for Life (BDFL)... but he's retiring!
- `http://www.artima.com/intv/` (search for Guido)

How Python is Managed

- Guido collects and writes Python Enhancement Proposals (PEPs)
- `http://www.python.org/dev/peps/`
- Interested parties comment
- Guido makes final decisions
- Team of programmers (many of them volunteers!) implement the features
- They're at version 3.0, but we're using version 2.5

Python as a Calculator

- The easiest way to experiment with Python is to enter statements into the Python shell
- Python supports many of the mathematical operators we're used to
- e.g. 11+56 at the shell gives 67
- Operators include + (addition), - (subtraction), * (multiplication), ** (exponentiation), / (division), % (remainder)

Python as a Calculator...

- Consider `7 / 3`
- The answer is 2? What's going on?
- Since the operands to / are integers (whole numbers), Python does integer division
- To get "real" division, at least one of the operators has to be a "floating-point" number (a number with a decimal point)
- e.g. `7.0 / 3.0`
- Floating-point numbers are sometimes only approximations!

Python Types

- Every Python value has a type that describes what sort of value it is, and its allowable operations
- The function `type` will tell us the type of any expression
- Python's types include
  - `int`: integer (no fractional part)
  - `long`: an integer that is too big or small for `int`
  - `float`: a number with a fractional component
- An expression involving values of the same type produces a value of that same type
- If one operand is a `float`, the other operand is converted up to a `float`, and so the result of the operation is a `float`

More Math

- Precedence: consider these two examples
  - `4 + 5 * 3`
  - `(4 + 5) * 3`
- Python uses the same precedence rules we're used to
- We can override these precedence rules with parentheses
- Python errors
  - `4 / 0`
  - `3 +`

Variables

- A variable is a name and an associated value
- Variables let us hang on to values so we can use them in several places
- In some ways, variables in programming are like math variables
  - e.g. in math, you could say "let $x = 5$"
  - Then, what is the value of $x * 3$?
- But, they are also different
  - e.g. in math, variables' values cannot change once they are given a value
  - In programming, we can change a variable as often as we like

Assignment Statement

- The assignment statement lets us give a value to a variable
- Form: `variable = expression`
- Two steps:
  1. Evaluate the expression on the right-hand side
  2. Associate the result with the variable on the left-hand side

Variable Examples

... various examples in the shell

- There is a difference between computing with a variable and assigning a new value to the variable
- You can't declare a permanent relationship between variables (but we will expand on this later!); e.g.

```
x = 37
y = x + 2
x
y
x = 20
y
```

Functions

- In math, we could define a function such as $f(x) = x^2$
- What is the value of $f(3)$? $f(5)$?
- In Python, we can achieve a similar effect

```
def f(x):
    return x ** 2
```

Now, what does the call `f (3)` do?

Functions...

- `def` is a keyword; it has a special meaning to Python and cannot be used as a variable or function name
- The `return` statement terminates the function, and determines the value returned to the caller
- If there is no `return`, the function terminates when it reaches the bottom (and returns `None`)
- Notice how the body of the function is indented! This is required by Python!
- We call `x` the **parameter** of function `f`
- `x` is a variable accessible only from within the function
- We could use any name we like for the parameter and/or function name

Functions...

- There is an important difference between a function definition and a function call!
- When we define a function with `def`, Python records it but does not execute it
- When we call a function, Python jumps to the first line of the function, executes it, and returns to the place where the function was called
- The purpose of functions is to define them once and call them whenever we want to use them
- To break up a complex calculation, we can use local variables inside functions
- Like parameters, they exist only during function execution; the variables do not exist outside of the function

Example: Function with Local Variables (func_localvars.py)

```
def polynomial(a, b, c, x):
    first = a * x * x
    second = b * x
    third = c
    return first + second + third
```

Names

- Naming rules and conventions apply to variable names, function names, and every other name you'll see
- They must begin with a letter or underscore (a-z, A-Z, _)
- Are made up of letters, underscores, and numbers
- Valid: `_moo_cow`, `cep3`, `I_LIKE_TRASH`
- Invalid: `49ers`, `@home`

Names...

- thEreS a GoOD rEasON wHy WorDs haVE A StaNDaRd caPITaLizAtIon sCHemE
- Python convention: pothole_case (e.g. `my_number`)
- CamelCase is sometimes seen, but not for functions and variables
- Rarely, single-letter names are capitalized: `L`, `X`, `Y`

Clickers: Functions (func_clicker.py)

Consider the following function definition:

```
def strange (x, y):
    z = 4
    z = z + x - y
    return z + 2
```

What is the value of `strange (6,1)`?

- A. 5
- B. 11
- C. 10
- D. 9
- E. 2

Intro to Media Module

- So far, we've only used built-in Python features
- A lot of what Python can do exists in external modules
- For example, the `media` module we will use shortly gives us access to sound and picture features
- It is not a standard Python module; it was created by U of T students!
- To use a module that is not built-in, we "import" it
- `import media`

Intro to Media Module...

- A basic feature of `media` is to let us display pictures
- The function `media.load_picture (filename)` lets us load a picture stored in a file
- It returns a `picture` object that we can store in a variable
- Then, the function `media.show` can be used to display the picture we have loaded
- For example, the following lines show the image in `bahen.jpg`

```
pic = media.load_picture ('bahen.jpg')
media.show (pic)
```

Digital Pictures

- Digital images are made up of pixels, which are tiny dots.
- That's what 1024 x 768 resolution means: 1024 pixels wide, 768 pixels high
- Pixel (0, 0) is upper left
- Pixel (1023, 0) is upper right
- Pixel (0, 767) is lower left
- Pixel (1023, 767) is lower right

Digital Pictures...

- Colours: combinations of red, green, and blue
- Each part has intensity in range 0 - 255
- Red: (255, 0, 0)
- Green: (0, 255, 0)
- Blue: (0, 0, 255)
- White: (255, 255, 255)
- Black: (0, 0, 0)

Picture at Sunset

- Our task: write a function to make a picture taken during the day look like it was taken at sunset
- Design: how can we accomplish this?
  - One way: go through the pixels one by one, and decrease the blue and green components of each
  - In pseudo-Python:

```
For each pixel in pic:
    get blue component of the pixel
    reduce that number by 30%
    set blue component of the pixel to the reduced number

    get green component of the pixel
    reduce that number by 30%
    set green component of the pixel to the reduced number
```

Picture at Sunset...

- Code: how can we convert this into Python?
- We can use the `for` statement
- In the case of pictures, it iterates through their pixels
- We should include the code in a function that takes a `Picture` object as parameter
- Our function will then make the picture look like it was taken at sunset
- We can then display the picture to see the changes

Sunset Function (sunset.py)

```
import media

def make_sunset(pic):

    for pixel in pic:

        # get pixel's old green value
        green = media.get_green(pixel)

        # set pixel to new green value
        media.set_green(pixel, int(green * 0.7))

        # get pixel's old blue value
        blue = media.get_blue(pixel)

        # set pixel to new blue value
        media.set_blue(pixel, int(blue * 0.7))
```

Design, Code, Verify

- A general methodology: Design, Code, Verify
- Design: determine the approach; reduce blue and green
- Code: write the code
- Verify: test it by running the code to see if it works as expected

Variable Scope (same_var.py)

What does this program do?

```
def f():
    x = 5
    print x

x = 9
print x
f()
print x
```

Variable Scope...

- The two `x` variables (in the main program and in function `f`) are independent!
- After we call function `f`, the value of `x` in the main program remains the same
- The `x` in the function is just a local variable! We know about these already
- Try as I might, the output of this program may not be obvious
- Another way to visualize the execution is through a debugger

Debugging

- So far, we have been using the Python shell
- Wing is a Python IDE, and it includes a debugger (as well as access to the shell we have been using)
- Two debugging commands
  - Step into (F7): executes one line of code, entering a function's code if one is called
  - Step over (F6): executes one line of code, or executes the entire body of a function in one step if one is called
- Let's trace our variable scope program in the debugger ...

Running Python Code

- There are lots of ways to run Python code
- From Wing: press F5 (or click Run)
- From the commandline: type `python filename.py`
- From the shell: type `import module_name`

Running Python Code...

- As we know, `import` is used to import a module
- Any file ending in `.py` can act as a module
- `import` runs all the code in a `.py` file, but only the first time (i.e. if you make changes to a file and try to import it again, nothing will happen)
- If we `import module_name`, we access its functions through `module_name.function_name`
- To access `function_name` directly, we can use `from module_name import function_name`

Running Python Code...

- If a file has code outside of a function (at the "top-level"), it will be executed when the code is `imported`
- This is usually not what we want, since an `import` means that other modules will use our functions
- When we run our code directly, though, we **do** wantthe top-level code to execute
- We can achieve this behavior by testing the `__name__` variable
- This is a common Python idiom: if you `import` a module, you get access to its functions; if you run it directly, the top-level code executes

Sample Module (first_mod.py)

```
def say_hi():
    print "Hi!"

def polynomial(a, b, c, x):
    first = a * x * x
    second = b * x
    third = c
    return first + second + third

if __name__ == "__main__":
    say_hi()
    print polynomial (2, 3, 4, 2)
```